Q 1

Overall chance of failure

$$= 1 - \prod_{i=1}^{n} \left[1 - (1 - r_i)^{b_i} \right]$$

To minimize the chance of failure, we need to maximize the second term.

To find b_i for each subsystem so that the plant remains functional but also under the budget B

Let A[i] be the array to denote maximum value of $\prod_{i=1} \left[1 - (1 - r_i)^{b_i} \right]$

when ~~out~~ our budget is i

∴ We get the recurrance formula

$$A[i] = \max_{1 \le k \le n} \left(A[i - c_k] \times \frac{1 - (1 - r_k)^{b_k + 1}}{1 - (1 - r_k)^{b_k}} \right)$$

when $i - c_k \geq 0$ & b_k = no. of subsystems

Q 2

We can calculate the problem by counting the possible number of step combination steps possible from the start node to every other node and saving it in an array.

i.e.,

if it's ~~takes~~ n combinations are possible to reach a cell (p, q), it there are n+1 possible combinations to reach

Any cell in the middle can be reached by the cell to its left & above, so the possible ~~count~~ combination at that cell will be the sum of those 2 cells.

Also, the base case will be at the beginning when there is only one possible action to the adjacent cells to the start node.

Pseudocode →

```
pathCount (m, n)
{
    for i in range(m)
        while  i<m
                count[i][0] = 1
    for j in range (0,n)
        while  j<n
                count[0][j] = 1

    for i in range(1, m):
        ~~for j in range(1,n):~~
        while  i <m :
            for j in range(1,n):
                while  j <n :
                    count[i][j] = count[i-1][j]
                                + count[i][j-1]
    return count[m-1][n-1]
}
```

Assignment 10

Q3) Since we have items in the same order when we sort by increasing weight or decreasing value, & the order of items which we have is arranged in ~~increasing~~ decreasing value/weight ratio

~~Ac~~ Following the greedy strategy, we can select the elements in the given order because we want to maximize the value, while minimizing the weight. This can be met by just accepting the items in the given order

Correctness:-

Assuming that there exists an item 'j' such that the item 'j' replaces one of the other items which come before it in the sorted order in an optimal set solution i.e. $j > i$

Since $j > i$,

$$weight_j > weight_i$$

. Having any other item while skipping a previous item means we ~~have~~ fill our knapsack with more weight. This will work if the value also increases

However, since we know the order is in decreasing value,

value$_j$ < value$_i$

. We are decreasing our value while also increasing our weight.

∴ This cannot be the optimal sdution.

∴ The proposed algorithm of selecting the items in the given order is optimal.

Q4.

We iterate through all elements in the array one by one while maintaining a variable maxJump which stores the maximum index we can jump to so far ~~out~~ while traversing the array.

Now, ~~if~~ after ~~iteration~~ traversal, if we reach the last element, we return true since we can reach the last element

However, during traversal, if the current index becomes more than the index maintained by maxJump, that means that we've come to an index which cannot be reached and thus, no further indexes can be reached.

maxJump should always be greater than the index of current array element being considered.

Pseudo code: →

```
reachEnd (A)
{
   initialize i = 0, maxJump = 0
   while (i < A.length and i <= maxJump)
    {
        maxJump = max(i + A[i], maxJump)
        i++
    }
   if  i == A.length
            return true
   else
            return false
}
```